FPGAで学ぶVerilog HDL



第3回　180秒のタイマを作る

デバイスの記事　ビギナーズ　関連データ

井倉将実

前回(2005年10月号，pp.130-134)は，1秒ごとにLEDが点灯/消灯を繰り返す(点滅する)回路を設計しました．この回路で「1秒ごとに動作する回路」を作ることができたわけです．その応用として，今回は台所で使うキッチン・タイマを作ってみることにしましょう．なお，本記事で紹介する設計データは本誌Webサイト(http://www.cqpub.co.jp/dwm/)からダウンロードすることができます．（筆者）

前回までで1秒ごとに動作する回路を設計できました．そこで最終的に時計を設計することを考えていきましょう．1秒ごとの動作からいきなり完全な時計を作るのはまだ壁が高そうです．そこで今回は，3分間クッキング用の180秒キッチン・タイマを作ります．

1.　180秒をカウントする

このキッチン・タイマは，180秒という値があらかじめセットされているものです．また，残り時間を3けたの7セグメントLEDに表示します．そしてスタート・ボタンを押すと1秒ごとにLEDの数字が減少し，180秒後には内部カウンタは0秒になります．このとき人間の目では，始まりの180から，179，178，177，…，000と，1秒ごとに減少するように見えます．0秒になると7セグメントLEDに「－」を点滅し，180秒が経過したことを示します．タイマ動作中にリセット・ボタンを押すと，タイマの値が180秒に戻り，次のスタート・ボタンが押されるまで待ち続けます．

図1に，設計するキッチン・タイマの動作をまとめます．**リスト1**は，180秒キッチン・タイマのVerilog HDL記述です．宣言部(モジュール宣言，ポート宣言，レジスタ宣言)が大きくなっていますが，行数が増えているだけで新しいことはありません．

記述は大きく五つの処理に分かれています．

- タイマ駆動開始フラグ生成
- 1秒タイミング生成
- 180秒タイマ回路
- タイムアウト検出回路
- 7セグメントLED表示部

● タイマ駆動開始フラグ生成――スイッチの処理

スイッチの押下状態によって，タイマのスタートと，タイマをリセットする機能を実現します．

ここでは，今回作る回路のいちばん重要な部分であるタイマ回路に対して適用される，二つのスイッチ状態に応じた「タイマ動作有効フラグ」を作っています(**図2**)．

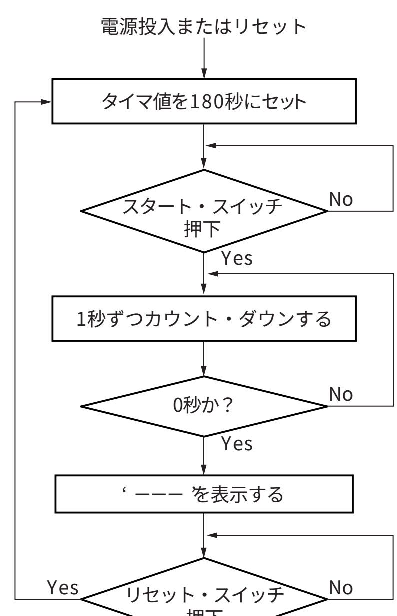


図1
設計するキッチン・タイマの動作
180秒という値があらかじめセットされている．スタート・ボタンを押すと1秒ごとにLEDの数字が減少し，0秒になると7セグメントLEDが「－」を点滅する．

タイマ動作有効フラグは，スタート・ボタンの押下によって有効（“H”）になり，リセット・ボタンの押下で無効（“L”）になります．タイマ回路には，このフラグが有効なときだけ動作させるようなしくみを持たせます．

ところで「スタート・ボタンを押したとき“H”，リセット・ボタンを押したとき“L”」という動作を単純に記述すると，**リスト2(a)**のようになります．しかし，この記述には落とし穴があります．スタート・ボタンとリセット・ボタンを同時に押したときに問題が起こるのです．

このような場合，どちらか一方のスイッチ入力を優先するように記述します．スイッチが同時に押されたときには，優先したスイッチの動作を実行します．

リセット・スイッチを優先した記述を**リスト2(b)**に示します．二つのスイッチが同時に押されたときには，タイマ動作有効フラグは無効（“L”）になります．

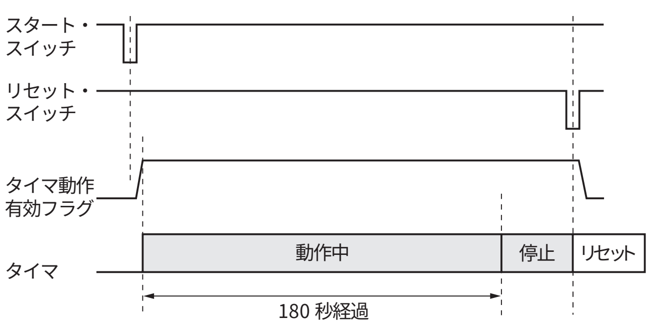


図2　タイマ動作有効フラグ

スタート・スイッチを押す（“L”になる）と，タイマ動作有効フラグが“H”になる．リセット・スイッチを押すと，タイマ動作有効フラグが“L”になる．

● 1秒タイミング生成回路

1秒タイミング生成回路は前回と同じ回路です．1秒ごとに1クロック分の幅だけ有効なフラグ信号（Pulse_1sec）が出力されます．

タイマ動作有効フラグ（TIM_START）が有効（“H”）のときのみ動作するような記述が追加されています．

リスト2　スタート・ボタンを押したとき“H”，リセット・ボタンを押したとき“L”にするVerilog HDL記述

```
always @( posedge SYSCLK )
begin
    if ( SW_START == 1'b0 ) begin
        TIM_START <= 1'b1 ;
    end

    if ( SW_RESET == 1'b0 ) begin
        TIM_START <= 1'b0 ;
    end
end
```

(a) 正しく動作しない記述

```
always @( posedge SYSCLK )
begin
    if ( SW_RESET == 1'b0 ) begin
                     // リセット・スイッチが押されたら
        TIM_START <= 1'b0 ;
                     //   -> タイマ駆動開始フラグをクリア
    end else begin   // リセット・スイッチが押されておらず
        if ( SW_START == 1'b0 ) begin
                     // スタート・スイッチが押されたら
            TIM_START <= 1'b1 ;
                     //   -> タイマ駆動開始フラグをセット
        end
    end
end
```

(b) リセット・スイッチを優先させる記述

リスト1　180秒キッチン・タイマのVerilog HDL記述

```
module TIM180(
// 入力ピン
    SYSCLK,
    SW_START,
    SW_RESET,

// 出力ピン
    SEG_1 ,
    SEG_2 ,
    SEG_3
);

// 入出力ピンの定義
    input               SYSCLK ;
    input               SW_START ;
    input               SW_RESET ;

    output     [7:0]    SEG_1 ;
    output     [7:0]    SEG_2 ;
    output     [7:0]    SEG_3 ;

    wire       [7:0]    SEG_1 ;
    wire       [7:0]    SEG_2 ;
    wire       [7:0]    SEG_3 ;

// 内部回路用のレジスタ宣言
    parameter  [31:0]   param_1Second = 32'h01F78A40 ;
                                // 33MHz時の1秒タイミング
    reg        [31:0]   Sec1_counter ;

    reg        [ 3:0]   TIM_3 ;       // 100秒けたタイマ
    reg        [ 3:0]   TIM_2 ;       //  10秒けたタイマ
    reg        [ 3:0]   TIM_1 ;       //   1秒けたタイマ

    reg        [ 7:0]   node_SEG3 ;   // 7Seg用内部ノード
    reg        [ 7:0]   node_SEG2 ;   // 7Seg用内部ノード
    reg        [ 7:0]   node_SEG1 ;   // 7Seg用内部ノード

    reg                 Pulse_1Sec ;
    reg                 TIM_START ;
    reg                 TIMEOUT ;

/*****************************************************/
// タイマ駆動開始フラグ生成
/*****************************************************/
always @( posedge SYSCLK )
begin
    if ( SW_RESET == 1'b0 ) begin
                              // リセット・スイッチが押されたら
```

```
        TIM_START <= 1'b0 ;
                        //   -> タイマ駆動開始フラグをクリア
    end else begin      // リセット・スイッチが押されておらず
        if ( SW_START == 1'b0 ) begin
                        // スタート・スイッチが押されたら
            TIM_START <= 1'b1 ;
                        //   -> タイマ駆動開始フラグをセット
        end
    end
end

/*****************************************************/
// 1秒タイミング生成
/*****************************************************/
always @( posedge SYSCLK or negedge SW_RESET )
begin
    if ( SW_RESET == 1'b0 ) begin
        Sec1_counter <= 32'b0 ;
        Pulse_1Sec      <= 1'b0 ;
    end else begin
        if ( TIM_START == 1'b1 ) begin
            if ( Sec1_counter == param_1Second ) begin
                Pulse_1Sec      <= 1'b1 ;
                Sec1_counter    <= 32'b0 ;
            end else begin
                Pulse_1Sec      <= 1'b0 ;
                Sec1_counter    <= Sec1_counter + 1 ;
            end
        end
    end
end

/*****************************************************/
// 180秒タイマ回路
/*****************************************************/
always @( posedge SYSCLK or negedge SW_RESET )
begin
    if ( SW_RESET == 1'b0 ) begin
        TIM_1 <= 4'b0000 ;                  // 180をセット
        TIM_2 <= 4'b1000 ;
        TIM_3 <= 4'b0001 ;
    end else begin
        if ( Pulse_1Sec == 1'b1 ) begin
            if ( TIMEOUT == 1'b1 ) begin
                              // もしタイムアウトを検出したら
                TIM_1 <= 4'b1111 ;        //   FFFをセット
                TIM_2 <= 4'b1111 ;
                TIM_3 <= 4'b1111 ;
            end else begin

            /***************/
            // 1秒カウンタ部
                if ( TIM_1 == 0 ) begin
                                  // 現在の値が0だったら
                    TIM_1 <= 4'b1001 ;//    9をセットする
                end else begin      // そうでなければ
                    TIM_1 <= TIM_1 - 1 ;
                                  //  カウントダウンする
                end

            /***************/
            // 10秒カウンタ部
                if ( TIM_1 == 0 ) begin// 1の位が0のとき
                    if ( TIM_2 == 0 ) begin
                                //   現在の値が0だったら
                        TIM_2 <= 4'b1001 ;
                                //9をセットする
                    end else begin //   そうでなければ
                        TIM_2 <= TIM_2 - 1 ;
                                //     カウントダウンする
                    end
                end

            /***************/
            // 100秒カウンタ部
               if ( ( TIM_1 == 0 ) & ( TIM_2 == 0 ) )
                     begin  // 1と10の位がともに0のとき
                    if ( TIM_3 == 0 ) begin
                             //   現在の値が0だったら
                        TIM_3 <= 4'b1001 ;//9をセットする
                    end else begin     //   そうでなければ
                        TIM_3 <= TIM_3 - 1 ;
                                  //カウントダウンする
                        end
                    end

                end
            end
        end
    end
end

/*****************************************************/
// タイムアウト検出回路
/*****************************************************/
always @( posedge SYSCLK or negedge SW_RESET )
begin
    if ( SW_RESET == 1'b0 ) begin
        TIMEOUT<= 1'b0 ;
    end else begin
        if ( ( TIM_3 == 4'b0000 ) &// 100秒の位がゼロ かつ
               ( TIM_2 == 4'b0000 ) &//  10秒の位がゼロ かつ
               ( TIM_1 == 4'b0000 )  //   1秒の位がゼロ
        ) begin
            TIMEOUT     <= 1'b1 ;
        end
    end
end


/*****************************************************/
// 7セグメントLED表示部
/*****************************************************/
always @( posedge SYSCLK )
begin
case ( TIM_3 )
    4'b0000 : node_SEG3 <= 8'b00111111 ;//'0' display
    4'b0001 : node_SEG3 <= 8'b00000110 ;//'1' display
    4'b0010 : node_SEG3 <= 8'b01011011 ;//'2' display
    4'b0011 : node_SEG3 <= 8'b01001111 ;//'3' display
    4'b0100 : node_SEG3 <= 8'b01100110 ;//'4' display
    4'b0101 : node_SEG3 <= 8'b01101101 ;//'5' display
    4'b0110 : node_SEG3 <= 8'b01111101 ;//'6' display
    4'b0111 : node_SEG3 <= 8'b00100111 ;/ '7' display
    4'b1000 : node_SEG3 <= 8'b01111111 ;//'8' display
    4'b1001 : node_SEG3 <= 8'b01101111 ;//'9' display
    default  : node_SEG3 <= 8'b01000000 ;//'-' display
endcase

case ( TIM_2 )
    4'b0000 : node_SEG2 <= 8'b00111111 ;//'0' display
    4'b0001 : node_SEG2 <= 8'b00000110 ;//'1' display
    4'b0010 : node_SEG2 <= 8'b01011011 ;//'2' display
    4'b0011 : node_SEG2 <= 8'b01001111 ;//'3' display
    4'b0100 : node_SEG2 <= 8'b01100110 ;//'4' display
    4'b0101 : node_SEG2 <= 8'b01101101 ;//'5' display
    4'b0110 : node_SEG2 <= 8'b01111101 ;//'6' display
    4'b0111 : node_SEG2 <= 8'b00100111 ;//'7' display
    4'b1000 : node_SEG2 <= 8'b01111111 ;//'8' display
    4'b1001 : node_SEG2 <= 8'b01101111 ;//'9' display
    default  : node_SEG2 <= 8'b01000000 ;//'-' display
endcase

case ( TIM_1 )
    4'b0000 : node_SEG1 <= 8'b00111111 ;//'0' display
    4'b0001 : node_SEG1 <= 8'b00000110 ;//'1' display
    4'b0010 : node_SEG1 <= 8'b01011011 ;//'2' display
    4'b0011 : node_SEG1 <= 8'b01001111 ;//'3' display
    4'b0100 : node_SEG1 <= 8'b01100110 ;//'4' display
    4'b0101 : node_SEG1 <= 8'b01101101 ;//'5' display
    4'b0110 : node_SEG1 <= 8'b01111101 ;//'6' display
    4'b0111 : node_SEG1 <= 8'b00100111 ;//'7' display
    4'b1000 : node_SEG1 <= 8'b01111111 ;//'8' display
    4'b1001 : node_SEG1 <= 8'b01101111 ;//'9' display
    default  : node_SEG1 <= 8'b01000000 ;//'-' display
endcase

end

/*****************************************************/
    assign     SEG_3[7:0]         = ~node_SEG3 ;
    assign     SEG_2[7:0]         = ~node_SEG2 ;
    assign     SEG_1[7:0]         = ~node_SEG1 ;

endmodule
```

● 180秒タイマ回路――タイマ動作中の処理

コードの先頭部には，タイマが動作しない状態のときの動作を記述しています．

- リセット・スイッチが押されたときは，カウンタに180をセットする．
- 1秒タイミング・パルス（Pulse_1sec）が有効（"H"）でタイムアウト・フラグ（TIMEOUT）が"H"のときは，カウンタの値を特別状態（FFF）にセットする．
- 1秒タイミング・パルス（Pulse_1sec）が有効（"H"）でタイマ動作有効フラグが"L"のときは，何もしない（それまでの値を保持し続ける）

タイマ回路は，以下の二つの条件がともに重なったときに動作（1秒ごとに減算）します．

- 1秒タイミング・パルス（Pulse_1sec）が有効（"H"）のとき
- タイマ動作有効フラグ（TIM_START）が有効（"H"）のとき

タイマ回路の内部は，100秒の位，10秒の位，1秒の位の三つのカウンタが入っており，自分よりも下の位の値を1秒ごとにチェックしてカウント動作を行います．

すなわち，1秒の位のカウンタは1秒ごとにつねに減算します．減算にあたっては，0だった場合には9をセットします．10秒の位のカウンタは，1秒の位のカウンタの値が0だった場合に，現在の値から1を減算します．100秒の位のカウンタは，10秒の位と1秒の位が00だった場合に，現在の値から1を減算します．

この動作で正しくカウントダウンできることを具体的に確認してみます．最初の値は180です．タイマがスタートして初めての1秒パルスが届いた瞬間を考えます．1秒の位はひたすら減算し続けるだけですが，0だったので9をセットします．10秒の位のカウンタは，1の位のカウンタが0だったため，1を減算し7になります．100秒の位のカウンタは，10秒の位のカウンタと1秒の位のカウンタがともに0という条件に一致しないので1を保持します．したがって，カウンタ値は179になります．

残り100秒の状態を考えます．1秒の位は減算により9がセットされます．10秒の位のカウンタは，1の位のカウンタが0だったため1を減算し，9がセットされます．下2けたが00であるため，100秒の位のカウンタは減算を行い0になります．結果として，内部カウンタ値は099となります．

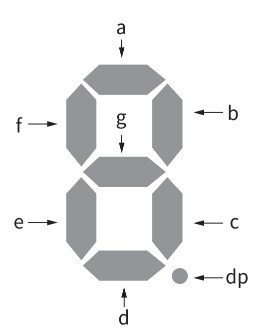
わたしたちが，ある数字から－1を行うときとまったく同じ計算方法により，カウントダウンを行っていることがわかります．カウンタ値が000になれば180秒が経過したことになります．

100の位の動作を見ると，000のあとは999に戻ってしまうことになります．カウンタの動作としてはこのようなしくみをとるのが一般的です．しかし，これだといつまでたってもごはんを食べられません（!?）．

そこで，000になったことを判断してタイマの動作を停止する機能が必要になります．

● タイムアウト検出回路――180秒経過を判定する処理

タイムアウト検出回路は，カウンタが動作し始めてから



（a）7セグメントLEDとビット位置



（b）7セグメント上の文字表示

図3
7セグメントLED
7個のLEDエレメントを使って数字や記号を表示できる部品である．小数点（dp）のエレメントを持つものも多い．（b）は筆者がよく利用する表示パターン．

180秒後を検出し，次に何を行うかを決める回路です．

180秒後の秒カウンタの値は，オール・ゼロ(000)です．タイムアウト検出回路は，1秒ごとに3けたの秒カウンタの値を監視し続け，オール・ゼロになったときにタイムアウト・フラグを出力します．タイムアウト・フラグは，いったんセットされると，リセット・スイッチが押されるまで“H”レベルを保ち続けます．

● 7セグメントLEDパネルの表示処理

FPGA内部でタイマを作っても，人の目にはその情報が伝わりません．そこで，3けたの7セグメントLEDを活用します．

7セグメントLEDは，**図3**にあるように，a～gの7個のLEDエレメントをどのように点灯するかによって，数字や記号を表示できる部品です．

リスト1を見ると，ずいぶんたくさんの記述があり，難しそうに感じるかもしれません．しかし，これは3けた分の表示を行うためであり，ほとんど同じ記述が三つあるだけです．1けた分の表示を行うのは，case-end caseで囲まれた部分のみです．

case文は，「入力に応じて出力を決定する」記述です．case文の記述を**図4**に示します．

case文では，case(nnnn)のカッコ内の信号を入力値として扱って，when xxxx : yyyyで入力信号がxxxxのときにyyyyという出力を行う構文です．when部は，入力信号の幅の組み合わせ個数分だけ書くことができます．例えば，4ビットの組み合わせであれば最大16個というわけです．そして，default部は，when部で書いた組み合わせ以外の入力条件の場合に出力される部分です．

キッチン・タイマでは，10進数しか表示しません．しかし，4ビットの信号には16種類の状態があります．そこで，実際に使う0～9以外の値が入力された場合の表示をdefaultで決めています．10～15の各状態についてすべて記述してもかまいませんが，defaultを使ったほうが簡単です．

```
case ( 入力信号 )
        when 組み合わせ状態 : 出力信号 = 論理式 ;
              :
        default :  出力信号 = 論理式 ;
end case ;
```

図4　case文の記述

case文は，あるまとまったビット幅を持つ信号の状態によって，出力状態を決定する場合に使います．if文で記述する場合と比べてソース・コードが読みやすくなり，まちがいを減らすことができます．

case文はalways文の中でだけ使うことができます．case文の出力信号はレジスタ型にします．入力信号はネット型とレジスタ型のどちらでも使うことができます．

● 7セグメントLEDへの出力

最後に，ネット型で宣言されている7セグメントLEDの最終出力信号に，生成した値の反転値を出力します．反転値(~；チルダと呼ぶ)を使う理由は，使用するFPGAボード(CQ出版の「FPGAボードで学ぶ論理回路設計」に付属するボード)の7セグメントLEDの各エレメントが電気的には“L”レベルで点灯，“H”レベルで消灯だからです．

● 動作確認

米国Altera社のQuartus IIを使って，論理合成と配置配線を行います．ピン配置は，**表1**のようになります．

表1　ピン配置

ピンの名まえ	入出力	接　続	ピン番号
SEG_3 [0]	出力	7セグメントLED (D3) のdp	21
SEG_3 [1]	出力	7セグメントLED (D3) のg	16
SEG_3 [2]	出力	7セグメントLED (D3) のf	15
SEG_3 [3]	出力	7セグメントLED (D3) のe	20
SEG_3 [4]	出力	7セグメントLED (D3) のd	22
SEG_3 [5]	出力	7セグメントLED (D3) のc	19
SEG_3 [6]	出力	7セグメントLED (D3) のb	14
SEG_3 [7]	出力	7セグメントLED (D3) のa	13
SEG_2 [0]	出力	7セグメントLED (D2) のdp	8
SEG_2 [1]	出力	7セグメントLED (D2) のg	5
SEG_2 [2]	出力	7セグメントLED (D2) のf	99
SEG_2 [3]	出力	7セグメントLED (D2) のe	7
SEG_2 [4]	出力	7セグメントLED (D2) のd	9
SEG_2 [5]	出力	7セグメントLED (D2) のc	6
SEG_2 [6]	出力	7セグメントLED (D2) のb	98
SEG_2 [7]	出力	7セグメントLED (D2) のa	97
SEG_1 [0]	出力	7セグメントLED (D1) のdp	94
SEG_1 [1]	出力	7セグメントLED (D1) のg	81
SEG_1 [2]	出力	7セグメントLED (D1) のf	87
SEG_1 [3]	出力	7セグメントLED (D1) のe	93
SEG_1 [4]	出力	7セグメントLED (D1) のd	96
SEG_1 [5]	出力	7セグメントLED (D1) のc	86
SEG_1 [6]	出力	7セグメントLED (D1) のb	85
SEG_1 [7]	出力	7セグメントLED (D1) のa	84
RESET	入力	スイッチ (SW2)	33
SW_START	入力	スイッチ (SW1)	34
CLK	入力	クロック発振器	39

リスト3　case文とif文の比較

```
case ( IN_SIGNAL )
    2'b00 : OUT_SIGNAL <= 4'b0001 ;
    2'b01 : OUT_SIGNAL <= 4'b0010 ;
    2'b10 : OUT_SIGNAL <= 4'b0100 ;
    2'b11 : OUT_SIGNAL <= 4'b1000 ;
endcase
```

(**a**) case文による記述

```
if ( IN_SIGNAL == 2'b00 ) begin
    OUT_SIGNAL <= 4'b0001 ;
end else begin
    if ( IN_SIGNAL == 2'b01 ) begin
        OUT_SIGNAL <= 4'b0010 ;
    end else begin
        if ( IN_SIGNAL == 2'b10 ) begin
            OUT_SIGNAL <= 4'b0100 ;
        end else begin
            if ( IN_SIGNAL == 2'b11 ) begin
                OUT_SIGNAL <= 4'b1000 ;
            end
        end
    end
end
```

(**b**) if文による記述

2. 設計のポイントのまとめ

● リセット状態について

今回は，リセット・スイッチにより，回路を初期状態にする記述を追加しています．

1秒タイミング生成回路のカウンタ値は，リセット直後にはオール・ゼロを示し，スタート・ボタンが押されることを待ち続けています．180秒という値を数えるキッチン・タイマの状態は，1秒ごとのフラグ信号が入力されていないためにカウント操作を行わず，その結果，180秒を示し続けてタイマの動作はストップします．

一般に，回路設計では，リセットによって初期状態に戻るしくみを組み込む必要があります．

● 回路全体はクロックで動作

LSIの回路は，クロック信号をもとに動作します．ソース・コードを見るとわかりますが，ほとんどすべての回路が，

```
always@( posedge CLK )  〜  begin  〜  end
```

という記述になっています．

この記述により，クロックの立ち上がりエッジごとに回路内部に書かれた記述どおりに動作することになります．

● イベントを認識する回路

前回の1秒ごとにLEDが点滅を繰り返すという回路は，クロックの入力ごとに自立して動いていました．それに対して今回の回路はスイッチの入力に応じてタイマを動かしたり，リセットと停止を行ったりと，操作要求に従って回路を駆動します．

このような回路の組みかたを「イベント・ドリブン型」の回路と言います．

今回のタイマの例では，スタート・スイッチとリセット・スイッチの二つの操作状況に応じて内部回路を動かしました．回路はこのスイッチを押下するという「イベント」に応じて動作するというわけです．

スタート・スイッチが押されてからリセット・スイッチが押されるまでの間に有効となるタイマ動作有効フラグは，イベント・ドリブン型の回路の基本となるので，今一度見直してみてください．

● case文とif文

7セグメント表示回路にはcase文を使っています．case文は，パターン化された入力データの羅列から任意の出力パターンを作るときによく使います．if文の塊で同じ機能を記述することもできますが，ソース・コードの見やすさという点ではcase文が勝ります．**リスト3**は，2ビットのネット型の入力信号から出力パターンを作るコードを，if分とcase文で記述したものです．**リスト3(b)**のケースは，if-begin-end-else-begin-if…となり，わかりにくくなります．

いくら・まさみ
FPGA-Magician

<筆者プロフィール>
井倉将実．本原稿執筆中にタイのタオ島で念願のダイビング免許を取得しました．機材の使いかたも専門用語もわからないヨチヨチ歩き状態ですが，Verilog HDL言語習得と同じように着実に身につけていこうと思っています．